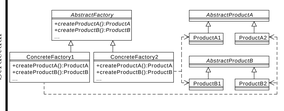
http://www.4shared.com/document/PrNfHQLE/posterSoftwareOntwerp.html

Design Patterns

Willem M. A. Van Onsem
Katholieke Universiteit Leuven
Academiejaar 2010-2011

Abstract Factory (Kit)

Structuur



Concept: Bieden van een interface voor het creëren van een families van gerelateerde of afhankelijke objecten zonder hun specifieke klassen te specificeren.

Deelnemers

- ***AbstractFactory:*** Definieert een interface voor operaties die abstracte producten creëren.
- ***ConcreteFactory:*** Implementeert de operaties om concrete producten te creëren.
- ***AbstractProduct:*** Definieert een interface voor een soort van producten.
- ***ConcreteProduct:*** Implementeert de abstracte product interface.

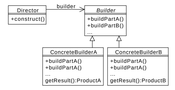
Gevolgen

- Isoleren van concrete klassen.
- Makkelijk veranderen van product-familie.
- Promoten van consistentie tussen verschillende producten.
- Ondersteuning voor nieuwe types producten is moeilijk.

Zie ook: *Factory Method, Prototype, Singleton*

Builder

Structuur



Concept: De constructie en representatie van een complex object scheiden, zodat hetzelfde constructieproces andere representaties kan creëren.

Deelnemers

- ***Builder:*** Definieert een interface voor deeloperaties bij de creatie van een product.
- ***ConcreteBuilder:*** Construeert en voegt delen samen door de interface te implementeren, en houd het object in aanbouw bij.
- ***Director:*** Bouwt object met de **Builder**.
- ***Product:*** Representeert een complex object onder constructie, inclusief klassen die als bouwblokken fungeren.

Gevolgen

- Makkelijk veranderen van de interne voorstelling van het product.
- Isoleert constructie- en voorstellingscode van elkaar.
- Geeft meer controle over de constructie van een object.

Zie ook: *Abstract Factory, Composite*

Factory Method

Structuur

Concept: Een interface definieren voor de bouw van een object, waarbij de subklasses het type object specificeren. *Factory Method* laat een klasse verschillende typen objecten bouwen.

Deelnemers

- ***Product:*** Definieert een algemene interface voor de producten die de *Factory Method* creëert.
- ***ConcreteProduct:*** Implementeert de **Product** interface.
- ***Creator:*** Declareert de *Factory Method* die een nieuw **Product** teruggeeft, eventueel met een standaard implementatie.
- ***ConcreteCreator:*** Overschrijft de Creator om een **ConcreteProduct** te bouwen.

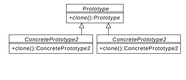
Gevolgen

- Meer flexibiliteit bij de constructie van subklassen.
- Verbindt parallelle product hiërarchieën.

Zie ook: *Abstract Factory, Prototype, Template Method*

Prototype

Structuur



Concept: Specificeren van het type objecten om te creëren met een prototype-object. Nieuwe objecten komen tot stand door de prototypes te kopëren.

Deelnemers

- ***Prototype:*** Definieert een interface om zichzelf te kunnen klonen.
- ***ConcretePrototype:*** Implementeert de **Prototype** interface door een kloon van zichzelf terug te geven.

Gevolgen

- Nieuwe soorten producten toevoegen en verwijderen tijdens uitvoering.
- Nieuwe objecten specificeren door het veranderen van de eigenschappen.
- Nieuwe objecten specificeren door het veranderen van de structuur.
- Vermindert subclassing (geen parallelle hiërarchieën zoals bij *Factory Method*).
- Dynamisch toepassingen configureren met klasse-injecties.

Zie ook: *Abstract Factory, Composite, Decorator*

Singleton

Structuur

Concept: Verzekeren van het ontstaan van slechts één instantie, en zorgen voor een globaal toegangspunt naar dit object.

Deelnemers

- ***Singleton:*** Biedt een statische methode die toegang verschaft tot de instantie. Verder is de **Singleton** ook verantwoordelijk voor de bouw van de instantie. En definieert ze een interface van methodes voor deze instantie.

Gevolgen

- Gecontroleerde toegang tot de enige instantie.
- Vermindert het aantal globale variabelen.
- Laat toe om operaties en voorstellingen verder te verfijnen in subklassen.
- Laat een variabel aantal instanties toe.
- Meer flexibel dan statische methodes.

Zie ook: *Abstract Factory, Builder, Prototype*

Chain of Responsibility

Structuur

Concept: Vermijden van het koppelen van de zender van een vraag en de ontvanger, door meer dan één object de kans te geven de vraag te beantwoorden. Op die manier beweegt de vraag zich doorheen de keten tot een object deze behandelt.

Deelnemers

- ***Handler:*** Definieert een interface om aanvragen te behandelen. Verder houdt het ook een referentie bij naar een eventuele opvolger.
- ***ConcreteHandler:*** Behandelt een aanvraag, verder kan het een beroep doen op zijn opvolger indien hij niet in staat is het verzoek af te handelen.

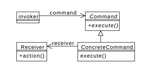
Gevolgen

- Verlaagt de koppeling.
- Verhoogt de flexibiliteit om verantwoordelijkheden aan objecten toe te kennen.
- Geen gegarandeerde afhandeling.

Zie ook: *Composite*

Command (Action)

Structuur



Concept: Encapsuleren van een vraag in een object, hierdoor kunnen opdrachten geparameteriseerd, bijgehouden, ongedaan gemaakt,... worden.

Deelnemers

- ***Command:*** Biedt een interface voor het uitvoeren van een operatie.
- ***ConcreteCommand:*** Implementeert het commando door de nodige operaties en de **Receiver** aan te roepen. Hierdoor fungeert het als een brug tussen de actie en zijn **Receiver** object.
- ***Invoker:*** Roept het commando op.
- ***Receiver:*** Weet hoe operaties moeten uitgevoerd worden bij het uitvoeren van een aanvraag.

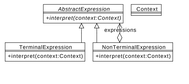
Gevolgen

- De **Command** klasse ontkoppelt de klasse die het commando oproept en de klasse die weet hoe dit commando moet worden uitgevoerd.
- **Commando** objecten kunnen aangepast en uitgebreid worden.
- Het is mogelijk **Command** objecten samen te voegen in een *Composite* patroon.
- Makkelijk om nieuwe commando's toe te voegen (klasses moeten niet aangepast worden).

Zie ook: *Prototype, Memento, Composite*

Interpreter

Structuur



Concept: Voor een gegeven taal, een representatie van de grammatica definiëren samen met een *Interpreter*, deze interpreteert zinnen in de gegeven taal.

Deelnemers

- ***AbstractExpression:*** Definieert een abstracte `interpret` operatie die alle objecten gemeen hebben in de syntax-boom.
- ***TerminalExpression:*** Implementeert de `interpret` operatie voor terminale symbolen in de grammatica.
- ***NonTerminalExpression:*** Houdt de kinderen bij die gegenereerd worden uit de expressie, en implementeert de `interpret` methode zodat deze de structuur uit een expressie kan afgeleid worden.
- ***Context:*** Houdt globale informatie bij voor de *Interpreter*.

Gevolgen

- Makkelijk om de grammatica uit te breiden of te veranderen.
- Implementeren van de grammatica is eenvoudig.
- Complexe grammatica's zijn moeilijk te onderhouden.
- Makkelijk om expressies op een andere manier te interpreteren.

Zie ook: *Iterator, Visitor, Composite, Flyweight*

Iterator (Cursor)

Structuur

Concept: Aanbieden van een manier om sequentieel toegang te verkrijgen tot objecten van een aggregaat-object. Dit zonder de details van het aggregaat-object te kennen.

Deelnemers

- ***Iterator:*** Definieert een interface om de lijst met objecten te kunnen overlopen.
- ***ConcreteIterator:*** Implementeert de **Iterator** interface, en houdt de huidige positie bij, bij het overlopen van het **Aggregate** object.
- ***Aggregate:*** Definieert een interface voor het creëren van een **Iterator**.
- ***ConcreteAggregate:*** Implementeert de creatie van de bijbehorende **ConcreteIterator**.

Gevolgen

- Ondersteund variatie in het doorlopen van een **Aggregate**.
- Een **Iterator** versimpelt de interface van een **Aggregate**.
- Verschillende **Iterator** objecten kunnen tegelijk een **Aggregate** doorlopen.

Zie ook: *Factory Method, Memento, Composite*

Mediator

Structuur

Concept: Definieren van een object die abstractie maakt over hoe een set van objecten interageren. Dit weerhoud objecten ervan expliciet naar elkaar te verwijzen, en laat toe de interactie dynamisch te wijzigen.

Deelnemers

- ***Mediator:*** Definieert een interface om te kunnen communiceren met **Collegue** objecten.
- ***ConcreteMediator:*** Implementeert coöperatief gedrag door **Collegue** objecten te coördineren.
- ***Collegue:*** Definieert een interface om met de **Mediator** te kunnen communiceren, en voorziet hierbij een referentie naar de **Mediator**.
- ***ConcreteCollegue:*** Communiceert telkens met het **Mediator** object, wanneer het dit zou doen met een ander object.

Gevolgen

- Het reduceert subclassing.
- Het gaat het koppelen van **Collegue** objecten tegen.
- Het versimpelt objectprotocollen.
- Het abstraheert hoe objecten met elkaar interageren.
- Het centraliseert de controle.

Zie ook: *Observer, Facade*

Memento (Token)

Structuur

Concept: Zonder de encapsulatie tegen te gaan het bekomen en opslaan van de interne toestand van een object. Dit object kan dan opnieuw zijn later toestand herstellen.

Deelnemers

- ***Memento:*** Slaat de interne toestand van een **Originator** op. Een **Memento** object wordt bijgehouden door de **Caretaker**.
- ***Originator:*** Bouwt **Memento** objecten als een momentopname van zijn huidige toestand. Het kan dan zijn interne toestand herstellen met behulp van een **Memento** object.
- ***Caretaker:*** Houdt de **Memento** objecten bij, maar raadpleegt nooit hun inhoud.

Gevolgen

- De **Memento** klasse kan de informatie afschermen van andere objecten.
- Het maakt de **Originator** klasse simpeler.
- *Memento* gebruikt veel geheugen en bewerkingen (kopiëren).
- Soms moeilijk om informatie af te schermen.
- Verborgen kosten in het beheren van **Memento** objecten.

Zie ook: *Command, Iterator*

Observer (Dependents)

Structuur

Concept: Definieren van een één-naar-veel afhankelijkheid tussen objecten, zodat indien het object van toestand verandert, al zijn afhankelijken verwittigd worden, en zich bijwerken.

Deelnemers

- ***Subject:*** Kent zijn **Observer** objecten, en biedt een interface om **Observer** objecten toe te voegen of te verwijderen.
- ***ConcreteSubject:*** Brengt de **Observer** op de hoogte van een veranderde toestand.
- ***Observer:*** Definieert een update interface dat aangeroepen moet worden, bij veranderingen van het **Subject** object.
- ***ConcreteObserver:*** Houdt een referentie bij naar het **ConcreteSubject** object. En implementeert de update interface die de interne toestand actueel houdt.

Gevolgen

- Abstracte koppeling tussen de **Subject** en **Observer**.
- Ondersteund mogelijkheden "omroep"-communicatie.
- Onverwachte updates leiden soms tot overhead en kettingreacties.

Zie ook: *Singleton, Mediator*

State (Objects for State)

Structuur

Concept: Een object toelaten zijn gedrag te wijzigen indien zijn interne toestand verandert. Het object zal in dat geval van klasse veranderen.

Deelnemers

- ***Context:*** Definieert een interface waarin de gebruiker geinteresseerd is. En houdt een referentie bij naar de huidige toestand.
- ***State:*** Bied een interface die gedrag – eigen aan een bepaald **State** object – definieert van het **Context** object.
- ***ConcreteState:*** Implementeert het gedrag specifiek voor deze toestand.

Gevolgen

- Encapsulatie van toestandsgebonden en opgesplitst gedrag.
- Het maakt toestandsovergangen explicieter.
- **State** objecten kunnen gedeeld worden, en de overhead reduceren.

Zie ook: *Singleton, Flyweight*

Strategy (Policy)

Structuur

Concept: Definieren en encapsuleren van van een familie van algoritmen die uitwisselbaar zijn. Een *Strategy* laat toe om dynamisch van algoritme te wijzigen onafhankelijk van de gebruikers.

Deelnemers

- ***Strategy:*** Definieert een interface die geldt voor alle ondersteunde algoritmen. De **Context** gebruikt deze interface om het algoritme in de **ConcreteStrategy** aan te roepen.
- ***ConcreteStrategy:*** Implementeert het algoritme gebruik makend van de interface van de **Strategy** klasse.
- ***Context:*** Definieert een interface waarin de gebruiker geinteresseerd is. En houdt een referentie bij naar de op dat moment gebruikte strategie. Verder biedt het vaak een interface zodat het **Strategy** object toegang heeft tot data.

Gevolgen

- Promoot families van gerelateerde algoritmen die hergebruikt kunnen worden.
- Biedt een dynamische alternatief voor subclassing.
- **Strategy** objecten elimineren conditionele statements.
- Dynamisch aanbieden van een implementatie-keuze.
- Gebruikers moeten zich bewust zijn van de verschillende strategieën.
- Communicatie-overhead tussen de **Strategy** en de **Context**.
- Verhoogt het aantal instanties in het systeem.

Zie ook: *Flyweight*

Template Method

Structuur

Concept: Definiëren van het skelet van een algoritme door operaties. Deze operaties worden dan op het niveau van de subklassen ingevuld. Dit laat subklassen toe een deel van het algoritme te wijzigen, zonder de structuur te veranderen.

Deelnemers

- ***AbstractClass:*** Definieert het skelet van een operatie waarbij het oproepen doet naar zowel concrete als abstracte operaties.
- ***ConcreteClass:*** Implementeert de abstracte operaties van de **AbstractClass** die specifiek zijn aan de klasse.

Gevolgen

- Promoot hergebruik van code.
- *"Hollywood-principe"*: omgekeerde controlestructuur: ouder roept de kinderen op, niet omgekeerd.

Zie ook: *Factory Method, Strategy*

Visitor

Structuur

Concept: Vertegenwoordigen van een operaties die toegepast moet worden op de objecten van een zekere structuur. Een *Visitor* laat toe de operatie te wijzigen zonder klassen van de objecten te wijzigen.

Deelnemers

- ***Visitor:*** Definieert een **Visitor** operatie voor iedere **ConcreteElement** klasse. De operatie-signatuur is specifiek voor het type element.
- ***ConcreteVisitor:*** Implementeert iedere methode voorzien door de **Visitor** vaak wordt hierbij een interne toestand aangepast.
- ***Element:*** Definieert een **accept** operatie die een specifieke **Visitor** accepteert.
- ***ConcreteElement:*** Implementeert de **accept** operatie en roept de operatie van de **Visitor** op.

Gevolgen

- Eenvoudig om nieuwe operaties toe te voegen.
- Een **Visitor** object verzamelt gerelateerde operaties, en ongerelateerde worden opgesplitst (in verschillende subklassen).
- Nieuwe **ConcreteElement** klassen toevoegen vraagt veel werk.
- Laat toe om een lijst van verschillende types objecten te doorlopen.
- Een **Visitor** kan een geaccumuleerde toestand bijhouden, wat tot minder doorgeven van parameters leidt.
- Verzwakt beschermingsniveaus: Een **Visitor** moet voldoende toegangsrechten hebben tegenover het **ConcreteElement**.

Zie ook: *Interpreter, Composite*

Adapter (Wrapper)

Structuur

Concept: Omzetten van de interface van een klasse in een andere interface die de gebruiker verwacht. Een *Adapter* laat twee klassen met elkaar samenwerken die dit anders niet kunnen omwille van incompatibele interfaces.

Deelnemers

- ***Target:*** Definieert een domein-specifieke interface die de gebruiker gebruikt.
- ***Adaptee:*** Definieert een bestaande interface die aangepast moet worden.
- ***Adapter:*** Past de **Adaptee** interface aan, aan de **Target** interface.

Gevolgen

- Aantal **Adaptee** objecten kan uitgebreid worden.
- Uitbreidbaar naar twee richtingen.
- Inplugbare adapters.

Zie ook: *Bridge, Decorator, Proxy*

Bridge (Handle; Body)

Structuur

Concept: Ontkoppelen van een abstractie en zijn implementatie zodat beide onafhankelijk kunnen variëren.

Deelnemers

- ***Abstraction:*** Definieert de abstractie interface. En houdt een referentie bij naar een **Implementor** object.
- ***RefinedAbstraction:*** Breidt de **Abstraction** interface uit.
- ***Implementor:*** Definieert een interface voor een **Implementor**. Deze interface is typisch primitiever dan de **Abstraction** interface.
- ***ConcreteImplementor:*** Implementeert de **Implementor** interface en definieert een concrete implementatie.

Gevolgen

- Ontkoppelen van interface en implementatie.
- Verbeterde uitbreidbaarheid.
- Implementatiedetails verbergen voor gebruikers.

Zie ook: *Abstract Factory, Prototype, Adapter*

Composite

Structuur

Concept: Schikken van objecten in een boom-structuur bij deel-geheel hiërarchieën. Dit laat toe om zowel individuele als compositie-objecten uniform te behandelen.

Deelnemers

- ***Component:*** Definieert een interface voor objecten in de compositie, deze bevat onder meer methodes die de kinderen van een object kunnen manipuleren. Implementeert standaard implementaties.
- ***Leaf:*** Implementeert operaties voor primitieve objecten, zonder kinderen.
- ***Composite:*** Definieert operaties voor objecten met kinderen, en slaat deze kinderen op. Operaties worden meestal doorgegeven naar de kinderen.

Gevolgen

- Recursief effect: geen transparante weergave.
- Vereenvoudigt de klasses die de *Composite* gebruiken.
- Makkelijk nieuwe types componenten toevoegen.
- Minder makkelijk om beperkingen op te leggen op de subklassen.

Zie ook: *Chain of Responsibility, Iterator, Visitor, Decorator, Flyweight*

Decorator (Wrapper)

Structuur

Concept: Voeg dynamisch nieuwe verantwoordelijkheden aan een object toe. Een *Decorator* biedt een flexibel alternatief tegenover subclassing voor uitgebreide functionaliteit.

Deelnemers

- ***Component:*** Definieert een interface voor objecten waar verantwoordelijkheden dynamisch aan kunnen toegevoegd worden.
- ***ConcreteComponent:*** Definieert een object waaraan extra verantwoordelijkheden kunnen worden toegevoegd.
- ***Decorator:*** Houdt een referentie naar een **Component** bij. En definieert een interface conform de interface van de **Component**.
- ***ConcreteDecorator:*** Voegt een verantwoordelijkheid toe aan het **Component**.

Gevolgen

- Meer flexibiliteit dan bij klassovererving.
- Vermijden van hoog verantwoordelijke klasses.
- Een **Component** en zijn **Decorator** zijn niet identiek.
- Veel kleine objecten.

Zie ook: *Strategy, Adapter, Composite*

Facade

Structuur

Concept: Aanbieden van een algemene interface voor een set van interfaces in een subsysteem. Een *Facade* definieert een interface van hoger niveau om een subsysteem eenvoudiger te kunnen gebruiken.

Deelnemers

- ***Facade:*** Weet welke klasses welke verantwoordelijkheid hebben in het subsysteem. En leidt aanvragen om naar de juiste objecten in het subsysteem.
- ***Subsystem Classes:*** Implementeert subsysteem functionaliteiten. En voert taken uit die door het **Facade** object worden aangevraagd, zonder weet van dit object te hebben.

Gevolgen

- Schermt de gebruiker af van het subsysteem, waardoor de gebruiker met minder objecten moet omgaan.
- Promoot zwakke koppeling tussen de gebruiker en het subsysteem.
- Houdt toepassingen niet tegen toch rechtstreeks het subsysteem te gebruiken.

Zie ook: *Abstract Factory, Singleton, Mediator*

Flyweight

Structuur

Concept: Deel objecten binnen een systeem, zodat grote hoeveelheden objecten efficiënt gebruikt kunnen worden, en duplicaten vermeden worden.

Deelnemers

- ***Flyweight:*** Definieert een interface voor **Flyweight** objecten.
- ***ConcreteFlyweight:*** Implementeert de **Flyweight** interface. Deze objecten worden gedeeld in de **FlyweightFactory**. Bijgevolg moet de interne toestand contextloos zijn.
- ***UnsharedConcreteFlyweight:*** Niet alle **Flyweight** objecten moeten gedeeld worden. Deze objecten implementeren de **Flyweight** interface.
- ***FlyweightFactory:*** Bouwt **Flyweight** objecten indien ze nog niet bestaan, anders wordt het gedeelde object teruggegeven.

Gevolgen

- Introduceert overhead bij het zoeken, en transfereren van objecten.
- Vermindert het aantal instanties die van een bepaalde klasse bestaan.
- Verlaagt het aantal inwendige toestanden van de objecten.

Zie ook: *State, Strategy, Composite*

Proxy (Surrogate)

Structuur

Concept: Een surrogaat-object om de toegang tot een andere object te controleren.

Deelnemers

- ***Proxy:*** Houdt een referentie bij, waardoor de **Proxy** een **RealSubject** kan aanspreken. Daarbij wordt een interface aangeboden die identiek is aan het **RealSubject** object. De **Proxy** controleert dan toegang tot dit object.
- ***Subject:*** Definieert een interface voor een **RealSubject** en **Proxy**. Hierdoor wordt de **Proxy** overal gebruikt waar men ook een **RealSubject** verwacht.
- ***RealSubject:*** Definieert het echte object die de **Proxy** vertegenwoordigt.

Gevolgen

- *Remote Proxy:* verbergt het feit dat de objecten ergens anders bestaan (op een andere computer).
- *Virtual Proxy:* kan informatie intern opslaan, en zo bijvoorbeeld oproepen uitstellen en optimaliseren (*Call-by-Need, Copy-on-Write* en *Creation-on-Demand* ↔ *Call-by-Value*).
- *Protection Proxy:* controleren van toegangsrechten alvorens de oproep uit te voeren.

Zie ook: *Adapter, Decorator*

Legende

- Creatie patroon
- Gedragspatroon
- Structureel patroon

KOMMUSOFT